Verification of machine learning based cyber-physical systems: a comparative study

Arthur Clavière
Collins Aerospace
Toulouse, France

Laura Altieri Sambartolomé
Collins Aerospace
Toulouse, France

Eric Asselin
Collins Aerospace
Toulouse, France

Christophe Garion
ISAE-SUPAERO
Toulouse, France

Claire Pagetti
ONERA
Toulouse, France

ABSTRACT

In this paper, we conduct a comparison of the existing formal methods for verifying the safety of cyber-physical systems with machine learning based controllers. We focus on a particular form of machine learning based controller, namely a classifier based on multiple neural networks, the architecture of which is particularly interesting for embedded applications. We compare both exact and approximate verification techniques, based on several real-world benchmarks such as a collision avoidance system for unmanned aerial vehicles.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Security and privacy** → **Logic and verification**.

KEYWORDS

Neural networks, Safety, Formal methods

ACM Reference Format:
Arthur Clavière, Laura Altieri Sambartolomé, Eric Asselin, Christophe Garion, and Claire Pagetti. 2022. Verification of machine learning based cyber-physical systems: a comparative study. In *25th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '22), May 4–6, 2022, Milan, Italy.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3501710.3519540

1 INTRODUCTION

In recent years, neural networks have emerged as a promising technology for the development of new avionic systems. They are a key enabler for advanced decision-making algorithms (*e.g.,* smart sensors), helping to provide enhanced performances and to reduce energy consumption [29]. However, their use in safety-critical systems (*e.g.,* aircraft) raises several issues in terms of confidence, performance, safety and certification. Indeed, in the worst case situation (*e.g.,* vision based applications), there is no proper definition for the expected behaviour of the neural network and thus no complete specification against which verifying and validating the system.

1.1 Context

In this paper, we do not tackle general machine learning based systems but we focus on *neural network based cyber-physical systems*, such as self-driving cars [8, 9, 23] or unmanned aerial vehicles [17]. For such systems, a system-level analysis is feasible. The idea is to model the behaviour of the whole system and to verify whether it enters in a non desired state or not. There are multiple papers and tools in the literature to propose *reachability analysis* [2, 3, 10]. The problem still remains complex as a neural network constitutes a highly non-linear, non-convex function, with high-dimensional input space and an exponentially large number of execution traces, making it difficult to analyze (verifying properties of a neural network is a NP-hard problem [18]).

In practice, the system is seen as the combination of a physical, continuous-time system with a discrete-time, neural network based controller. Thus, it can be modelled as a *hybrid system* [4]. Additionally, we focus on a particular type of neural network based controller, which we call a *classifier based on multiple networks*. The controller could be implemented as a unique neural network, but it can also be made of several neural networks with a switch between them.

1.2 Contributions

The purpose of the paper is to review the formal methods available in the literature to identify the most suited ones to tackle a given verification problem. For that, we have selected:

- 3 state-of-the-art tools, namely NNV [19, 26], VENMAS [1] and SAMBA [11].
- 3 benchmarks that are representative of the domain, namely the Vertical Collision Avoidance System (VCAS) [1, 16], the Airborne Collision Avoidance System (ACAS) [17] and the Cartpole [20].

The contributions of this paper are threefold. First, we have *formalized* the problem of ensuring the safety of a set of cyber-physical systems, each one equipped with a classifier based on multiple networks. In practice each system is abstracted as a hybrid automaton with a single location and a single transition. The dynamics within the location consists of an ordinary differential

https://doi.org/10.1145/3501710.3519540

equation. The overall system, composed of multiple cyber-physical systems, is the product of the hybrid automata, with random offset. To ensure the safety of this system, the verification problem consists in demonstrating that no dangerous state can be reached. For instance, for a drone fleet, we want to ensure that no collision can happen. Because the general problem is highly complex, we tackle – as everyone in the literature – a simpler problem where the cyber-physical systems are assumed to be synchronized. Indeed, this hypothesis reduces the exploration space.

The second contribution is a *thourough and fair comparison* between the verification tools. The comparison relies on the three selected use cases which present different types of dynamics, linear or non-linear, and involve neural networks of different sizes. The tools are compared based on their capacity to solve a large variety of verification problems.

The third contribution are the *lessons learned* from this comparison. We discuss the applicability of formal methods and we propose a verification strategy. We show that the verification effort mainly lies in the analysis of the dynamics of the physical system, the nature of which highly influences the performance of the different methods.

1.3 Organization

The paper is organized as follows. Section 2 describes our model of a machine learning based cyber physical system and defines the associated verification problem as a reachability problem. Section 3 presents the applicable formal verification techniques and section 4 describes our methodology for comparing these methods. Sections 5-7 describe the benchmarks and present the results of the experiments. Section 8 discusses the lessons learned from the experiments.

2 PROBLEM STATEMENT

2.1 Cyber-physical system with periodically scheduled controller

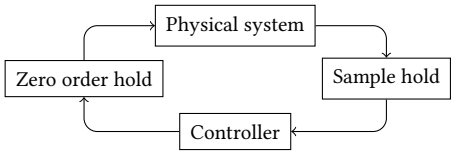


Figure 1: Block diagram representation of a CPS with periodically scheduled controller.

Let us consider a Cyber-Physical System (CPS). Such a system combines a physical system, with continuous dynamics, and a software controller, with discrete dynamics. Let us also assume that the software controller is executed *periodically*, with a period T. At the beginning of the period, the controller samples the state of the physical system. Then, based on this sample, it computes an actuation command for the physical system. This new actuation command is applied at the end of the period, allowing the controller to finish its execution at any point within the period. It is important to note that the physical system has continued to evolve between the beginning and the end of the period: when the command is applied, the state of the physical system has changed compared to the state sampled by the controller. This behaviour is equivalent to having a controller that is executed instantaneously, at the end of the period, and which takes as input a record of the state of the physical system at the beginning of the period.

A hybrid automaton (HA) is a common representation for this type of system, exhibiting both continuous and discrete dynamics. The general definition of a HA can be found in [4]. In order to model the CPS of interest, we consider a particular HA, showed in Figure 2 and described in definition 2.1.

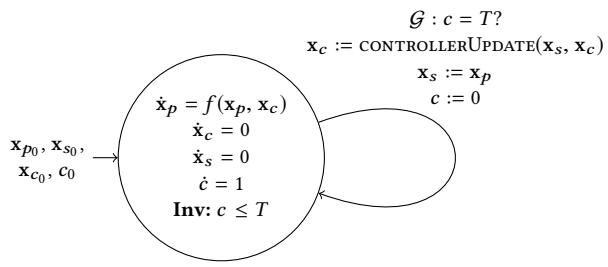


Figure 2: Hybrid automaton representation of a CPS with periodically-scheduled controller.

Definition 2.1. (Hybrid automaton representation of a CPS with periodically scheduled controller) [6] A hybrid automaton representing a CPS with periodically scheduled controller is a tuple $\mathcal{H} = (l, \mathbf{x}, Inv_l, Flow_l, \delta, X_0)$ where:

- l is the unique location, modelling the continuous dynamics of the physical system,
- $\mathbf{x} = (\mathbf{x}_p, \mathbf{x}_s, \mathbf{x}_c, c)$ are the variables: $\mathbf{x}_p \in \mathbb{R}^m$ is the state of the physical system; $\mathbf{x}_s \in \mathbb{R}^m$ and $\mathbf{x}_c \in \mathbb{R}^q$ are the variables of the controller, representing the sampled state of the physical system and the actuation command respectively; $c \in \mathbb{R}_{\geq 0}$ is a clock, used to ensure the periodic execution of the controller,
- Inv_l is the invariant in location l, defined as $Inv_l : c \leq T$. Indeed, within the location l, time elapses until the period T is reached *i.e.,* until the variables of the controller are updated,
- $Flow_l$ is a set of ordinary differential equations (ODEs) defining the time evolution of the variables in location l. The time evolution of the physical system depends on the actuation command from the controller. Hence the associated ODE $\dot{\mathbf{x}}_p = f(\mathbf{x}_p, \mathbf{x}_c)$ where $f : \mathbb{R}^m \times \mathbb{R}^q \to \mathbb{R}^m$ is a Lipschitz continuous function to ensure the existence and uniqueness of a solution [22]. The variables $\mathbf{x}_c$ and $\mathbf{x}_s$ are updated periodically and remain constant between two updates. Therefore, the associated ODEs are $\dot{\mathbf{x}}_c = 0$ and $\dot{\mathbf{x}}_s = 0$. The variable c is a clock with rate 1 hence $\dot{c} = 1$,
- $\delta = (l, \mathcal{G}, \mathcal{M}, l)$ is the unique discrete transition, from and to the location l. It represents the periodic update of the variables of the controller. The guard is defined as $\mathcal{G} : c = T?$. Regarding the update function $\mathcal{M}$, we want the actuation command to be updated based on the *previously* sampled

state of the physical system and the *previous* actuation command. A transition in a HA must not use any memory. Therefore, we circumvent the problem by using the variables $\mathbf{x}_s$ and $\mathbf{x}_c$ which remain constant between two transitions, allowing to keep track of the previously sampled state and the previous command respectively. The corresponding update function is $\mathbf{x}_c := \text{controllerUpdate}(\mathbf{x}_s, \mathbf{x}_c)$, where $\mathbf{x}_s$ and $\mathbf{x}_c$ are actually the previously sampled state and the previous command. Once the command is updated, $\mathbf{x}_s$ is assigned the value of the current state $\mathbf{x}_p$ and the clock is reseted to 0: $\mathbf{x}_s := \mathbf{x}_p$ and $c := 0$.

- X_0 is the set of the possible initial values of the variables. It is defined as $X_0 = \{(\mathbf{x}_{p_0}, \mathbf{x}_{c_0}, \mathbf{x}_{s_0}, c_0) \mid \mathbf{x}_{p_0} \in X_{p_0}, \mathbf{x}_{s_0} = \mathbf{x}_{p_0}, \mathbf{x}_{c_0} \in X_{c_0}, c_0 = 0\}$ where $X_{p_0} \subset \mathbb{R}^m$ is the set of the possible initial states of the physical system and $X_{c_0} \subset \mathbb{R}^q$ is the set of the possible initial actuation commands. We have $\mathbf{x}_{s_0} = \mathbf{x}_{p_0}$ since $\mathbf{x}_{s_0}$ is the sampled state of the physical system at instant 0, and $c_0 = 0$ since the clock initially equals 0.

2.2 Network of cyber-physical systems with periodically scheduled controller

The model of definition 2.1 represents a *single* CPS. If we are interested in representing a *network* of n CPSs (*e.g.,* a drone fleet), the corresponding model is the product of n hybrid automata of type $\mathcal{H}$ (see Figure 3).

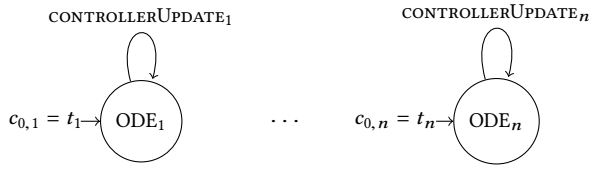


Figure 3: Model of a network of n CPSs.

In the general case, the start times $t_1, \ldots, t_n$ are *non-determistic*, making the analysis of the model quite complicated. To ease the reasoning about this model, we consider the following assumption, as all research papers do to our knowledge:

Assumption 1. *The controllers of the n CPSs are synchronized i.e.,* $t_1 = \ldots = t_n$.

Under assumption 1., the model becomes a single hybrid automaton, where the flow relation (*resp* the update function) is the composition of the ODEs (*resp* the update functions) of the n CPSs. The corresponding HA is showed in Figure 4.

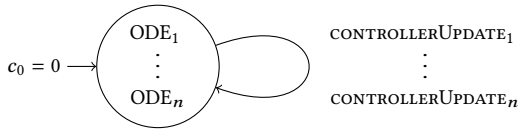


Figure 4: Model of a network of n CPSs with synchronized controllers.

2.3 Neural network based controller

This section focuses on the definition of the controllerUpdate function when the controller is a *classifier based on multiple networks*. Such a controller produces a command from a *finite* set of possible commands, hence the name *classifier*. Moreover, it disposes of several neural networks, among which only one is used at each execution. The neural network to be used is selected based on the previous actuation command.

Definition 2.2. (Classifier based on multiple networks) A classifier based on multiple networks is a controller defined by a tuple $(\mathcal{U}, \mathcal{N}, \lambda, Pre, Post)$ where:

- $\mathcal{U}$ is a *finite* set of actuation commands, the size of which is denoted by d *i.e.,* $\text{card}(\mathcal{U}) = d$,
- $\mathcal{N}$ is a finite set of deep feedforward ReLU neural networks. The networks in $\mathcal{N}$ all have the same input space and the same output space. The function computed by a network N in $\mathcal{N}$ is denoted by $F_N : \mathbb{R}^p \to \mathbb{R}^d$,
- $\lambda : \mathcal{U} \to \mathcal{N}$ maps every command in $\mathcal{U}$ to a neural network in $\mathcal{N}$,
- $Pre : \mathbb{R}^m \to \mathbb{R}^p$ is a pre-processing function (*e.g.,* normalization),
- $Post : \mathbb{R}^d \to \mathcal{U}$ is a post-processing function, defined as $Post(\mathbf{y}) = \pi(\text{argmax}_i\ \mathbf{y}_i)$ or $Post(\mathbf{y}) = \pi(\text{argmin}_i\ \mathbf{y}_i)$, where $\pi : [\![1, d]\!] \to \mathcal{U}$ is a bijection and $\mathbf{y}_i$ denotes the i^{th} component of vector $\mathbf{y}$, with $i \in [\![1, d]\!]$.

One execution of the controller consists of the following steps:

(i) it takes as inputs the previously sampled state $\mathbf{x}_s \in \mathbb{R}^m$ and the previous actuation command $\mathbf{x}_c$
(ii) it executes the *Pre* function, which transforms the sampled state $\mathbf{x}_s$ into a vector $\mathbf{z} \in \mathbb{R}^p$ *i.e.,* $\mathbf{z} = Pre(\mathbf{x}_s)$,
(iii) it selects the network N to be used, depending on the previous actuation command *i.e.,* $N = \lambda(\mathbf{x}_c)$,
(iv) it executes the selected network, which yields an output $\mathbf{y} \in \mathbb{R}^d$: $\mathbf{y} = F_N(\mathbf{z})$,
(v) it executes the *Post* function, which gives the new actuation command $\mathbf{x}_c = Post(\mathbf{y})$.

Overall, the corresponding function is defined as:

$$\text{controllerUpdate} \begin{cases} \mathbb{R}^m \times \mathcal{U} \to \mathcal{U} \\ (\mathbf{x}_s, \mathbf{x}_c) \mapsto Post \circ F_{\lambda(\mathbf{x}_c)} \circ Pre(\mathbf{x}_s) \end{cases} \quad (1)$$

2.4 Reachability problem

Let $\mathcal{H}_{NN}$ be a HA representing a CPS with periodically-scheduled controller and where the controller is a classifier based on multiple networks. Let us assume that the set of the possible initial states of the physical system X_{p_0} is a m-box *i.e.,* the cartesian product of m intervals, and that the set of the possible initial commands is $X_{c_0} = \{\mathbf{u}_0\}$ where $\mathbf{u}_0 \in \mathcal{U}$. Let us also consider a m-box E_p representing a set of unsafe states of the physical system, leading to a failure of the CPS *e.g.,* a collision in a drone fleet.

We are interested in deciding if $\mathcal{H}_{NN}$ has a safe behaviour, which we express as the following *reachability problem*: decide if $\mathcal{H}_{NN}$ can reach a state where $\mathbf{x}_p \in E_p$ within a bounded time horizon $t_{\text{end}} = KT$, where $K \in \mathbb{N}^*$.

To formalize this reachability problem, we need to describe the possible behaviours of $\mathcal{H}_{NN}$. Usually, the behaviours of a HA are

described as *runs*. The general definition of a run can be found in [7]. In the case of $\mathcal{H}_{NN}$, a run alternates between time intervals of length T, where the HA is in location l and the variables evolve according to the associated ODEs, and discrete transitions, which update the variables through the mapping $\mathcal{M}$.

Definition 2.3. (Run of $\mathcal{H}_{NN}$) A run of size K of the hybrid automaton $\mathcal{H}_{NN}$ is a sequence:

$$(l, \mathbf{x}_{p_0}, \mathbf{x}_{s_0}, \mathbf{x}_{c_0}) \xrightarrow{T} (l, \mathbf{x}_{p_1}, \mathbf{x}_{s_0}, \mathbf{x}_{c_0}) \xrightarrow{\delta} (l, \mathbf{x}_{p_1}, \mathbf{x}_{s_1}, \mathbf{x}_{c_1}) \\ \dots \xrightarrow{\delta} (l, \mathbf{x}_{p_K}, \mathbf{x}_{s_K}, \mathbf{x}_{c_K})$$

where:

- $\mathbf{x}_{p_0} \in X_{p_0}$ and $\mathbf{x}_{c_0} = \mathbf{u}_0$,
- for all $k \in [\![0, K]\!]$, $\mathbf{x}_{s_k} = \mathbf{x}_{p_k}$
- for all $k \in [\![0, K-1]\!]$, $\mathbf{x}_{p_{k+1}}$ is the value at $t = T$ of the solution of the ODE $\dot{\mathbf{x}}_p = f(\mathbf{x}_p, \mathbf{x}_{c_k})$ with the initial condition $\mathbf{x}_p(t = 0) = \mathbf{x}_{p_k}$,
- for all $k \in [\![0, K-1]\!]$, $\mathbf{x}_{c_{k+1}} = \text{CONTROLLERUPDATE}(\mathbf{x}_{s_k}, \mathbf{x}_{c_k})$

Based on the above definition, we define the reachability problem as follows:

Definition 2.4. (Reachability problem) The reachability problem consists in deciding if there is a run of $\mathcal{H}_{NN}$ of size K such that $(l, \mathbf{x}_{p_k}, \mathbf{x}_{s_k}, \mathbf{x}_{c_k})$ belongs to this run and $\mathbf{x}_{p_k} \in E_p$.

3 FORMAL VERIFICATION TECHNIQUES

Several techniques can handle the reachability problem described in Definition 2.4. In the following, we distinguish between *exact* and *approximate* verification techniques. Both provide a *sound* result: a *no* answer to the reachability problem means that $\mathcal{H}_{NN}$ actually has a safe behaviour *i.e.,* no state in E_p can be reached. However, exact techniques are *complete* while approximate techniques are not. Indeed, exact techniques answer either *yes* or *no* to the reachability problem, while approximate techniques cannot answer *yes* due to approximation.

3.1 Exact verification

An exact representation of the reachability problem can be constructed using Mixed Integer Linear Programming (MILP), under the following assumptions:

ASSUMPTION 2. *For all $\mathbf{x}_{p_{init}} \in \mathbb{R}^m, \mathbf{x}_c \in \mathcal{U}$, the unique solution of the ODE $\dot{\mathbf{x}}_p = f(\mathbf{x}_p, \mathbf{x}_c)$ with the initial condition $\mathbf{x}_p(t = 0) = \mathbf{x}_{p_{init}}$ is a function $t \mapsto g(\mathbf{x}_{p_{init}}, \mathbf{x}_c, t)$ where g is* linear or piecewise linear *in $\mathbf{x}_{p_{init}}$ and $\mathbf{x}_c$*

ASSUMPTION 3. *The Pre function of the controller is* linear or piecewise linear.

The MILP encoding of the reachability problem consists of a collection of K feasibility problems: $(\mathcal{P}_k)_{1 \leq k \leq K}$. The problem $\mathcal{P}_k$ is to decide if there is a run of size k of $\mathcal{H}_{NN}$ such that $\mathbf{x}_{p_k} \in E_p$, where $\mathbf{x}_{p_k}$ is the state of the physical system at the end of the run. More formally, given the definition of a run of $\mathcal{H}_{NN}$ and the above assumptions, the problem $\mathcal{P}_k$ is defined as follows:

$$\begin{aligned} &\text{find } \mathbf{x}_{p_j}, \mathbf{x}_{s_j}, \in \mathbb{R}^m, \mathbf{x}_{c_j} \in \mathcal{U} \\ &\text{s.t. } \mathbf{x}_{p_0} \in X_{p_0} \wedge \mathbf{x}_{c_0} = \mathbf{u_0} \\ &\quad \mathbf{x}_{p_k} \in E_p \\ &\quad \forall j \in [\![0, k]\!], \mathbf{x}_{s_j} = \mathbf{x}_{p_j} \\ &\quad \forall j \in [\![0, k-1]\!], \mathbf{x}_{p_{j+1}} = g(\mathbf{x}_{p_j}, \mathbf{x}_{c_j}, T) \\ &\quad \forall j \in [\![0, k-1]\!], \mathbf{x}_{c_{j+1}} = \text{CONTROLLERUPDATE}(\mathbf{x}_{s_j}, \mathbf{x}_{c_j}) \end{aligned} \tag{2}$$

where

- $\mathbf{x}_{c_{j+1}} = \text{CONTROLLERUPDATE}(\mathbf{x}_{s_j}, \mathbf{x}_{c_j})$ can be encoded as MILP constraints since CONTROLLERUPDATE is piecewise linear in $\mathbf{x}_{s_j}$ and $\mathbf{x}_{c_j}$: linear constraints can be used for the linear components of the function (*e.g.,* the connections between the layers of the neural networks) and big-M encoding can be used for the piecewise linear components of the function (*e.g.,* the ReLU activation functions in the neural networks),
- $\mathbf{x}_{p_0} \in X_{p_0}$ and $\mathbf{x}_{p_k} \in E_p$ can be encoded as MILP constraints since X_{p_0} and E_p both consist of m-boxes,
- $\mathbf{x}_{p_{j+1}} = g(\mathbf{x}_{p_j}, \mathbf{x}_{c_j}, T)$ can be encoded as MILP constraints since g is linear or piecewise linear in $\mathbf{x}_{p_j}$ and $\mathbf{x}_{c_j}$.

If there is $k^* \in [\![1, K]\!]$ such that $\mathcal{P}_{k^*}$ is feasible, then $\mathcal{H}_{NN}$ can reach a state in E_p and the answer to the reachability problem is *yes*. Otherwise, the answer is *no*: $\mathcal{H}_{NN}$ has a safe behaviour.

A similar MILP approach was implemented in the VENMAS tool [1], which aims at verifying strategic properties of neural-symbolic multi-agent systems. In this tool, the choice of the actuation command is partially non-deterministic, which we modified to reproduce the deterministic behaviour described in section 2.3. Additionally, VENMAS assumes that the controller is executed instantaneously, which we did not modify. Practically, this makes the problem a bit simpler than the one described in equation (2): the decision variables $\mathbf{x}_{s_j}$ and the constraints $\mathbf{x}_{s_j} = \mathbf{x}_{p_j}$ are removed. Moreover, $\mathbf{x}_{s_j}$ is replaced by $\mathbf{x}_{p_j}$ in the constraints $\mathbf{x}_{c_{j+1}} = \text{CONTROLLERUPDATE}(\mathbf{x}_{s_j}, \mathbf{x}_{c_j})$.

3.2 Approximate verification

An approximate verification of the reachability problem can be performed by constructing an *over-approximating flowpipe*, containing all the possible runs of $\mathcal{H}_{NN}$ (and potentially more). Unlike exact verification, there is no restrictive assumption for constructing this flowpipe.

The over-approximating flowpipe consists of a *tree*, an example of which is showed in Figure 5. This tree can be built inductively, based on the definition of a run.

ALGORITHM 1. *(Reachability tree construction)* The construction of the reachability tree consists of the following steps:

step (i) The root of the tree is the tuple $(l, X_{p_0}, X_{s_0}, \mathbf{x}_{c_0})$ where $X_{s_0} = X_{p_0}$ and $\mathbf{x}_{c_0} = \mathbf{u}_0$,

step (ii) This root has a single child, $(l, X_{p_1}, X_{s_0}, \mathbf{x}_{c_0})$, where X_{p_1} is an over-approximation of the possible solutions at $t = T$ of the ODE $\dot{\mathbf{x}}_p = f(\mathbf{x}_p, \mathbf{x}_c)$, with the initial condition $\mathbf{x}_p(t = 0) \in X_{p_0}$. The node $(l, X_{p_1}, X_{s_0}, \mathbf{x}_{c_0})$ is thus an over-approximation of the reachable states of $\mathcal{H}_{NN}$ at the end of the first period T,

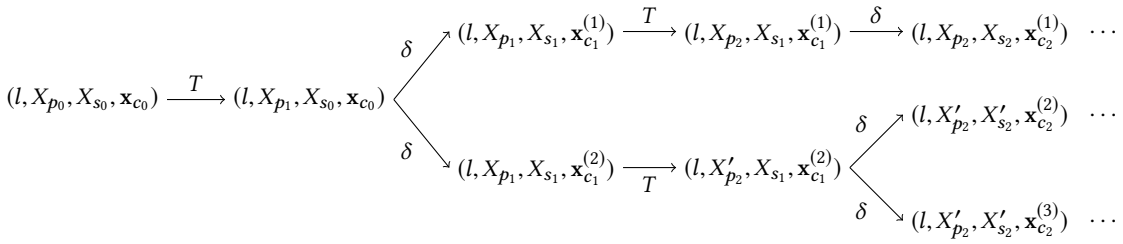


Figure 5: An example reachability tree of $\mathcal{H}_{NN}$.

step (iii) The node $(l, X_{p_1}, X_{s_0}, \mathbf{x}_{c_0})$ can have several children, namely $(l, X_{p_1}, X_{s_1}, \mathbf{x}_{c_1}^{(1)}), \ldots (l, X_{p_1}, X_{s_1}, \mathbf{x}_{c_1}^{(n)})$, representing the reachable states of $\mathcal{H}_{NN}$ after the first discrete transition. These children result from the update of $\mathbf{x}_s$ *i.e.,* $X_{s_1} = X_{p_1}$, and the propagation of $(X_{s_0}, \mathbf{x}_{c_0})$ through the controllerUpdate function. The set X_{s_0} is propagated through the *Pre* function, yielding a set $Z \subset \mathbb{R}^p$. Then, the network N to be executed is selected as $N = \lambda(\mathbf{x}_{c_0})$ and the set Z is propagated through the associated function F_N, yielding the set $Y \subset \mathbb{R}^d$. Given that Y is a set, its propagation through the argmax (or argmin) function involved in *Post* may not yield a unique element. We denote by $\mathbf{x}_{c_1}^{(1)}, \ldots, \mathbf{x}_{c_1}^{(n)}$ the possible actuation commands resulting from the propagation of Y through the *Post* function. As the actuation command drives the selection of the neural network at next execution, several paths are created, as many as the number of possible commands, hence the tree representation.

step (iv) Each node $(l, X_{p_1}, X_{s_1}, \mathbf{x}_{c_1}^{(1)}), \ldots (l, X_{p_1}, X_{s_1}, \mathbf{x}_{c_1}^{(n)})$ is considered as the root of a tree that is constructed by applying stages (ii) and (iii). This is repeated until the height of the node $(l, X_{p_0}, X_{s_0}, \mathbf{x}_{c_0})$ equals K *i.e.,* the time horizon t_{end} is reached.

If each node $(l, X_p, X_s, \mathbf{x}_c)$ in the tree satisfies $X_p \cap E_p = \emptyset$, then the answer to the reachablility problem is *no*: $\mathcal{H}_{NN}$ has a safe behaviour. Otherwise, due to the approximation process - we may not be able to calculate the exact set of the reachable solutions of the ODE nor the exact output set of the controllerUpdate function - one cannot conclude about the reachability problem.

To the best of our knowledge, there exist two complete implementations of this approach:

- a MATLAB framework [19, 21] based on the Neural Network Verification tool (NNV) [26], which we refer to as NNV for simplicity. Altough the code was originally dedicated to the analysis of the ACAS Xu system, we adapted it to handle any system of the type of $\mathcal{H}_{NN}$,
- the Safety Assessment of Machine learning Based Autonomous systems tool (SAMBA) [11].

There are other tools which can calculate an over-approximating flowpipe for CPSs with neural network based controllers [12, 14, 15, 28]. However, these tools assume that the controller is a unique neural network and do not tackle the case where the controller is a classifier based on multiple networks. There is also an ad hoc reachability method, dedicated to the verification of the VCAS and the ACAS systems [16], where the controller is a classifier based on multiple networks. This method, instead of constructing a reachability tree, explores the entire state space: it discretizes the state space and calculates all the possible behaviours of the controller in each of the resulting cells.

In the following, we focus on NNV and SAMBA. Both implement the algorithm given in definition 1. Steps (i) and (iv) do not require any specific implementation. For steps (ii) and (iii), they work as follows:

step (ii), in NNV, X_{p_0} is represented as a *zonotope i.e.,* a centrally symmetric convex polytope. The CORA tool [3] is used to approximate the reachable solutions of the ODE, yielding the zonotope X_{p_1}. In SAMBA, X_{p_0} is represented as a *m-box*. The DynIBEX tool [2] is used, yielding the m-box X_{p_1}.

Due to the use of zonotopes, NNV keeps track of the relations between the state variables, which SAMBA does not. Taking account of these relations makes the successive approximations of the dynamics more precise. However, it is more expensive, particularly for non-linear ODEs, due to the addition of noise symbols and new relations.

step (iii), in NNV and SAMBA, X_{s_0} is propagated through the controllerUpdate function by using *abstract interpretation*. To this end, the following *abstract transformers* are used: $Pre^{\#}$ and $Post^{\#}$ approximate the reachable outputs of the *Pre* and *Post* functions. They are based on *interval abstraction* in both NNV (more precisely our adaptation of NNV) and SAMBA. The abstract transformers $F_N^{\#}$, with $N \in \mathcal{N}$, approximate the reachable outputs of the neural networks. In NNV, they are based on *star set abstraction*, which is an effective representation of high-dimensional polytopes[5, 27]. In SAMBA, they are based either on *zonotope abstraction*, implemented by the DeepZono tool [13], or on a *mixed interval-polytope abstraction*, implemented by the DeepPoly tool [24].

In NNV and SAMBA, the abstract transformers $F_N^{\#}$ rely on an approximation of the ReLU function, defined as $\sigma(x) = \max(0, x)$. In particular, they propose an approximation in the pathological case where the two segments $\sigma(x) = 0$ and $\sigma(x) = x$ are reachable. These approximations are illustrated in Figure 6. One can notice that the star based approximation used in NNV is the most precise. Moreover, contrary to the star based and the polytope based approximations, the zonotope based approximation does not capture the fact that $\sigma(x) \geq 0$.

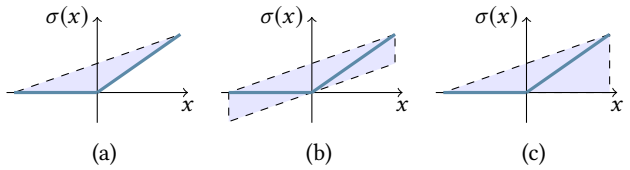


Figure 6: ReLU approximation with star set (NNV) (a), zonotope (SAMBA) (b) and polytope (SAMBA) (c) [27].

4 COMPARISON METHODOLOGY

To evaluate the verification methods against a large variety of verification problems, we considered four criteria:

Criterion #1: Use case We evaluated the tools against three use cases that fit into the model of section 2, namely the VCAS, the ACAS and the Cartpole. These use cases were chosen as they correspond to real-world systems and they have been widely studied in the literature [20]. Moreover, they present different degrees of complexity in terms of verification. The VCAS has a simple linear dynamics but it involves large-sized networks and it has many possible commands (card($\mathcal{U}$) = 81). The difficulty in verifying the VCAS lies in (1) the analysis of the neural networks and (2) the large number of constraints (for exact techniques) or the large size of the reachability tree (for approximate techniques) due to the large number of possible commands. The ACAS uses neural networks of similar size, but it has a more complex, non-linear dynamics and it has fewer possible commands. For the ACAS, the difficulty of the verification is more balanced between (1) the analysis of the neural networks and (2) the analysis of the dynamics. Finally, the Cartpole has a quite more complex, highly non-linear dynamics, but it has only 2 possible commands and it involves small neural networks. The difficulty of the verification lies essentially in the analysis of the dynamics. The characteristics of the different use cases are summarized in table 1.

	# of possible commands	size of the networks	dynamics
VCAS	81	large	linear
ACAS	25 or 5	large	non-linear
Cartpole	2	small	highly non-linear

Table 1: Comparison of the use cases.

Criterion #2: Type of the verification problem For these three use cases, we did not consider the entire set of the possible initial states. Indeed, due to the large size of this initial set, a complete verification would have been very costly [11, 16]. Moreover, such a verification may require a heuristic for partitionning the set of the possible initial states [11], which all tools do not implement. Instead, for each use case, we created a set of verification problems. A verification problem is a tuple $(\mathbf{x}_{p_0}, \mathbf{x}_{c_0}, \mathbf{unc}, t_{\text{end}})$ where $\mathbf{x}_{p_0} \in \mathbb{R}^m$ is an initial state of the physical system, $\mathbf{x}_{c_0} \in \mathbb{R}^q$ is an initial command, $\mathbf{unc} \in \mathbb{R}^m$ is an uncertainty on the initial state and t_{end} is a time horizon. It corresponds to the reachability problem defined in section 2.4 where the set X_{p_0} is the box with upper bound $\mathbf{x}_{c_0} + \mathbf{unc}$ and lower bound $\mathbf{x}_{c_0} - \mathbf{unc}$. Such a verification problem can be of different types, depending on the form of the uncertainty **unc**. Uncertainties on position variables are easy to propagate through the dynamics. Indeed, position variables do not drive the time evolution of any state variable. On the contrary, handling uncertainties on angle or velocity variables is more complex since such variables drive the time evolution of other state variables. Moreover, uncertainties on angle or velocity variables can make the set of the reachable states grow very fast, making the verification task harder. As a consequence, uncertainties on angle or velocity variables may demand more accuracy for the verification tools, especially for the analysis of the dynamics. Finally, uncertainties on all state variables require an accurate approximation of both the dynamics and the neural networks. They correspond to the most complex problems. In our experiments, we considered the three following types of problems:

problem type	description
A	uncertainties on position variables only
B	uncertainties on angle/velocity variables only
C	uncertainties on all state variables

Table 2: Types of verification problems.

Criterion #3: Time horizon the larger the time horizon t_{end} is, the more complex the verification problem becomes. Indeed, for exact techniques, increasing the time horizon adds a large number of decision variables and constraints. For approximate techniques, in case of a loose approximation, the repeted approximations may lead to a false negative: the over-approximation is too large to answer to the verification problem. A loose approximation can also lead to a large reachability tree: more commands are considered to be reachable, resulting in more branches in the tree and making the analysis increasingly expensive with larger time horizons. On the contrary, an accurate approximation can also result in a very expensive analysis, due to the repeted cost of a precise approximation. A large time horizon thus requires to provide a good balance between accuracy and scalability.

Criterion #4: Criticity of the verification problem the initial state $\mathbf{x}_{p_0}$ can be such that the verification problem is either *non-critical* or *critical*. Non-critical problems are such that, without any command from the controller *i.e.,* with $\mathbf{x}_{c_0}$ constantly equal to zero, the system will remain in a safe state for the time interval $[0, t_{\text{end}}]$. Critical problems are such that, without any command from the controller, the system will reach an unsafe state at $t = t_{\text{end}}$. Critical problems are expected to be more complex to solve and to require a certain level of accuracy for the verification tools. In the following we denote by $\mathcal{NC}$ a non-critical problem and by $\mathcal{C}$ a critical problem.

Experiments description For each use case, we considered problems of types A, B and C. We also considered three different time horizons and we created $n_{\mathcal{NC}}$ non-critical problems and $n_{\mathcal{C}}$ critical problems. Overall, we evaluated each use case on $3 \times 3 \times (n_{\mathcal{NC}} + n_{\mathcal{C}})$ problems. The results are shown in Table 3 which shows the percentage of problems solved versus the *cumulative* verification time, depending on the different criteria. As an example, a **100% / 52s** result in the column B means that all the problems of

type B could be solved in a total time of 52s. Additionally, figure 8 compiles the results for the totality of the problems, independently of the criteria. Note that we considered a timeout of 1h and that a timeout is considered as a failure to solve the verification problem.

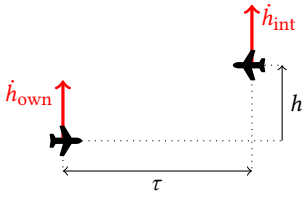
5 BENCHMARK 1: VERTICAL COLLISION AVOIDANCE SYSTEM (VCAS)

5.1 Description

The VCAS system consists of two aircraft, both equipped with a collision avoidance controller. This controller periodically provides a vertical maneuver advisory, based on a classifier with multiple networks. We assume that the two controllers are synchronized, so that the system can be represented by a single automaton, as explained in section 2.2. The verification problem consists in demonstrating that no collision can happen, starting from a given set of configurations.

Variables: if the first aircraft is called the *ownship* and the second aircraft is called the *intruder*, then the state of the physical system is the vector $\mathbf{x}_p = (h, \dot{h}_{\text{own}}, \dot{h}_{\text{int}}, \tau)$ where h is the altitude of intruder relative to ownship (in ft), $\dot{h}_{\text{own}}$ and $\dot{h}_{\text{int}}$ are the vertical rates of the two aircraft (in ft/s), and τ is the time before loss of horizontal separation between the two aircraft. The actuation command is the vector $\mathbf{x}_c = (x_{c_{\text{own}}}, x_{c_{\text{int}}})$ where $x_{c_{\text{own}}}$ and $x_{c_{\text{int}}}$ are the vertical accelerations of the ownship and the intruder respectively (in ft/s^2).

Flow relation: the ODE $\dot{\mathbf{x}}_p = f(\mathbf{x}_p, \mathbf{x}_c)$ is given in equation (3).



$$\begin{cases} \dot{h} = \dot{h}_{\text{int}} - \dot{h}_{\text{own}} \\ \ddot{h}_{\text{own}} = x_{c_{\text{own}}} \\ \ddot{h}_{\text{int}} = x_{c_{\text{int}}} \\ \dot{\tau} = -1 \end{cases} \tag{3}$$

Figure 7: Dynamics of the VCAS.

Controller of the ownship: the controller of the ownship has a period $T = 1s$. It produces a command among a set of 9 possible commands. The Pre_{own} is the identity function. The controller disposes of a collection of 9 neural networks $\mathcal{N} = \{N^{(1)}, \ldots, N^{(9)}\}$, available at [25]. These networks all have 5 hidden layers of 45 nodes each. They yield a 9 dimensional output, on which the function $Post_{\text{own}} : \mathbf{y} \mapsto \pi(\text{argmax}_i\ \mathbf{y}_i)$ is applied.

Controller of the intruder: the controller of the intruder is the same as the one of the ownship, except that Pre_{int} returns the vector $(-h, \dot{h}_{\text{int}}, \dot{h}_{\text{own}}, \tau)$, where $-h$ is the altitude of ownship relative to intruder. Indeed, the intruder observes the system from its own point of view.

Overall, the product of the two CPSs has $9^2 = 81$ possible commands.

It is worth noting that both the Assumptions 2. and 3. for using VENMAS are satisfied (see section 3.1). Indeed, the solution of the ODE (3) with the initial condition $\mathbf{x}_{p_0} = (h_{\text{own},0}, h_{\text{int},0}, \dot{h}_{\text{own},0}, \dot{h}_{\text{int},0}, \tau_0)$ and the initial command $\mathbf{x}_{c_0} = (x_{c_{\text{own},0}}, x_{c_{\text{int},0}})$ is the function $t \mapsto (h_0 - \dot{h}_{\text{own},0} \cdot t - 0.5 \cdot x_{c_{\text{own},0}} \cdot t^2 + \dot{h}_{\text{int},0} \cdot t + 0.5 \cdot x_{c_{\text{int},0}} \cdot t^2, \dot{h}_{\text{own},0} + x_{c_{\text{own},0}} \cdot t, \dot{h}_{\text{int},0} + x_{c_{\text{int},0}} \cdot t, \tau_0 - t)$, which is linear in $\mathbf{x}_{p_0}$ and $\mathbf{x}_{c_0}$. Moreover, Pre_{own} and Pre_{int} are linear functions.

Verification problems: the set of the unsafe states is $E_p = \{\mathbf{x}_p \mid |h| < 100.0 \ \wedge \ |\tau| < \epsilon\}$ *i.e.,* the cases where the two aircraft collide. The problems of type A consist of an uncertainty of ±5ft on h_0, the problems of type B consist of an uncertainty of ±1ft/s on $\dot{h}_{\text{int},0}$, and the problems of type C consist of an uncertainty of ±5ft on h_0 and ±1ft/s on $\dot{h}_{\text{int},0}$ and $\dot{h}_{\text{own},0}$. The time horizons are $t_{\text{end}} \in \{5s, 8s, 25s\}$. Finally, we created 2 non-crititical and 4 critical verification problems, yielding a total of 54 problems.

5.2 Results

Table 3 and page12 show the results. NNV and SAMBA (polytope) can solve all the verification problems while SAMBA (zonotope) cannot solve the problems of type C and VENMAS can only solve simple problems (type A or B and small t_{end}), often reaching timeouts for complex problems. In terms of performance, VENMAS is significantly slower than approximate methods: the large number of possible commands and the large size of the neural networks make the number of decision variables and constraints quite important. Regarding approximate techniques, SAMBA (polytope) always outperforms SAMBA (zonotope). SAMBA (polytope) is faster than NNV for simple problems (type A or B) while NNV outperforms SAMBA for complex problems (type C). Such problems are more costly, hence the overall better performance of NNV (see figure 8).

Interpretation. Compared to SAMBA (polytope and zonotope), NNV offers a more precise approximation of the neural networks, which is also faster in average. However, its approximation of the dynamics does not provide significantly tighter bounds and it is more costly. This higher cost is due to keeping track of the relations between the state variables and the tool used, which is different to SAMBA. Overall, the approximation of SAMBA is less costly but it is also less precise, especially for the networks. This leads to a larger reachability tree: some commands are considered to be reachable although they are not. This phenomenon is accentuated by the large number of possible commands. Indeed, in the worst case, the width of the reachability tree grows exponentially, at a rate equal to the number of possible commands. As a comparison, NNV constructs reachablity trees with at most 9 branches for the VCAS while SAMBA (polytope) builds trees with at most 707 branches. The balance between a costly approximation (NNV) and a large reachability tree (SAMBA) is in favor of SAMBA (polytope) for simple problems while it is in favor of NNV for more complex problems. Similarly, SAMBA (polytope) is faster than SAMBA (zonotope) because the zonotope approximation is less precise and leads to larger trees.

6 BENCHMARK 2: AIRBORNE COLLISION AVOIDANCE SYSTEM (ACAS)

6.1 Description

The ACAS benchmark is similar to the VCAS. It involves two aircraft: an ownship, equipped with a collision avoidance controller,

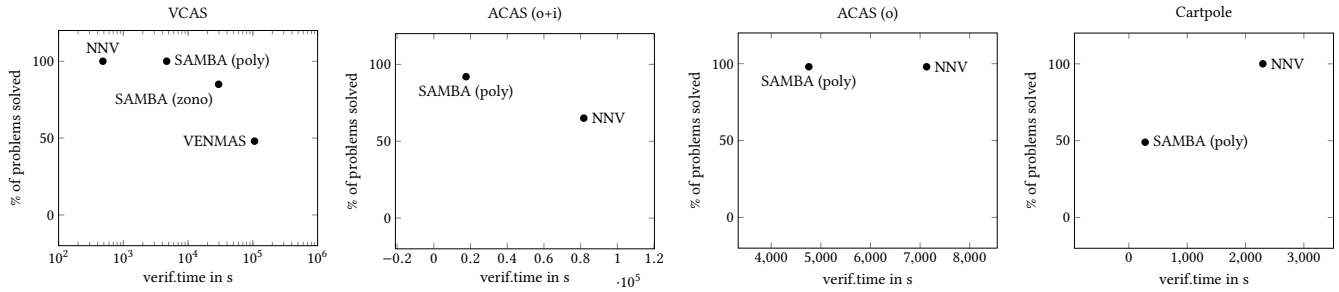


Figure 8: Performances of the verification tools. The best tool is at the top left hand corner.

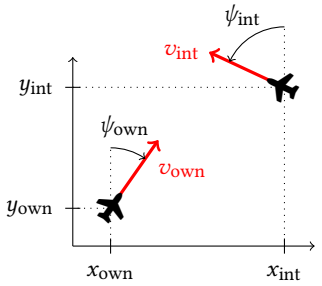
	VCAS benchmark							
	$\mathcal{NC}$ problems	$\mathcal{C}$ problems	$t_{\text{end}} = 5.0$	$t_{\text{end}} = 8.0$	$t_{\text{end}} = 25.0$	A	B	C
VENMAS	**44%** / 36347s	**50%** / 69843s	**94%** / 4947s	**44%** / 40043s	**6%** / 61200s	**61%** / 26913s	**56%** / 31291s	**28%** / 47985s
NNV	**100%** / 165s	**100%** / 318s	**100%** / 77s	**100%** / 104s	**100%** / 303s	**100%** / 154s	**100%** / 160s	**100%** / 170s
SAMBA (poly)	**100%** / 125s	**100%** / 4544s	**100%** / 80s	**100%** / 204s	**100%** / 4385s	**100%** / 29s	**100%** / 82s	**100%** / 4557s
SAMBA (zono)	**89%** / 7356s	**83%** / 22241s	**94%** / 3918s	**83%** / 11081s	**78%** / 14598s	**100%** / 45s	**100%** / 176s	**56%** / 29376s
	ACAS (o+i) benchmark							
	$\mathcal{NC}$ problems	$\mathcal{C}$ problems	$t_{\text{end}} = 16.0$	$t_{\text{end}} = 32.0$	$t_{\text{end}} = 56.0$	A	B	C
NNV	**67%** / 22137s	**64%** / 59567s	**71%** / 21891s	**43%** / 43485s	**81%** / 16329s	**90%** / 8530s	**76%** / 18715s	**29%** / 54459s
SAMBA (poly)	**100%** / 242s	**89%** / 17326s	**95%** / 802s	**95%** / 5452s	**86%** / 11314s	**100%** / 224s	**95%** / 5251s	**81%** / 12093s
	ACAS (o) benchmark							
	$\mathcal{NC}$ problems	$\mathcal{C}$ problems	$t_{\text{end}} = 16.0$	$t_{\text{end}} = 32.0$	$t_{\text{end}} = 56.0$	A	B	C
NNV	**100%** / 671s	**98%** / 6460s	**100%** / 318s	**100%** / 1230s	**95%** / 5583s	**100%** / 705s	**100%** / 1014s	**95%** / 5412s
SAMBA (poly)	**100%** / 214s	**98%** / 4544s	**100%** / 95s	**100%** / 357s	**95%** / 4306s	**100%** / 292s	**100%** / 315s	**95%** / 4151s
	Cartpole benchmark							
	$\mathcal{NC}$ problems	$\mathcal{C}$ problems	$t_{\text{end}} = 0.2$	$t_{\text{end}} = 0.5$	$t_{\text{end}} = 1.0$	A	B	C
NNV	**100%** / 1363s	**100%** / 934s	**100%** / 297s	**100%** / 561s	**100%** / 1439s	**100%** / 760s	**100%** / 773s	**100%** / 764s
SAMBA (poly)	**48%** / 170s	**50%** / 108s	**100%** / 31s	**40%** / 69s	**7%** / 179s	**47%** / 75s	**53%** / 48s	**47%** / 155s

Table 3: Performances of the tools, depending on the characteristics of the verification problems. Legend: a green cell indicates the best verification tool.

and an intruder, equipped *or not* with a collision avoidance controller. The case where both the ownship and the intruder are equipped with the controller is denoted (o+i) and the case where only the ownship has a controller is denoted (o). The case (o) is interesting: it may demand less computational efforts (the controller of the intruder is not executed) but a collision is more likely to happen, potentially requiring more precision to solve the reachability problem. Moreover, the case (o) does not require any synchronization hypothesis (only one controller is used).

Variables: for cases (o+i) and (o), the state of the physical system is the vector $\mathbf{x}_p = (x_{\text{own}}, y_{\text{own}}, x_{\text{int}}, y_{\text{int}}, v_{\text{own}}, v_{\text{int}}, \psi_{\text{own}}, \psi_{\text{int}})$ where $(x_{\text{own}}, y_{\text{own}})$ and $(x_{\text{int}}, y_{\text{int}})$ are the 2D cartesian coordinates of the two aircraft (in ft), v_{own} and v_{int} are their horizontal velocities (in ft/s), ψ_{own} and ψ_{int} are their heading angles (measured counter clockwise in rad). In the case (o+i), the actuation command is $\mathbf{x}_c = (x_{c_{\text{own}}}, x_{c_{\text{int}}})$ while in the case (o), the command is $\mathbf{x}_c = x_{c_{\text{own}}}$.

Flow relation: In the case (o+i), the ODE $\dot{\mathbf{x}}_p = f(\mathbf{x}_p, \mathbf{x}_c)$ is given in equation (4), where the two aircraft are assumed to have a constant horizontal velocity. In the case (o), the ODE is the same but $\mathbf{x}_{c_{\text{int}}} = 0$ is a constant: the intruder has a uniform rectilinear displacement.



$$\begin{cases} \dot{x}_{\text{own}} = -v_{\text{own}} \cdot \sin(\psi_{\text{own}}) \\ \dot{y}_{\text{own}} = v_{\text{own}} \cdot \cos(\psi_{\text{own}}) \\ \dot{x}_{\text{int}} = -v_{\text{int}} \cdot \sin(\psi_{\text{int}}) \\ \dot{y}_{\text{int}} = v_{\text{int}} \cdot \cos(\psi_{\text{int}}) \\ \dot{v}_{\text{own}} = 0 \\ \dot{v}_{\text{int}} = 0 \\ \dot{\psi}_{\text{own}} = x_{c_{\text{own}}} \\ \dot{\psi}_{\text{int}} = x_{c_{\text{int}}} \end{cases} \tag{4}$$

Figure 9: Dynamics of the ACAS.

Controller of the ownship: the controller of the ownship has a period $T = 1s$. It produces a command among a set of 5 possible commands. The Pre_{own} function returns the cylindrical coordinates of the intruder relative to ownship: it yields the vector $(\rho, \theta_{\text{int/own}}, v_{\text{own}}, v_{\text{int}}, \psi_{\text{int/own}})$ where ρ is the distance between the two aircraft, $\theta_{\text{int/own}}$ is the angle of the intruder relative to the ownship heading direction and $\psi_{\text{int/own}}$ is the heading angle of the intruder relative to the ownship heading direction. The controller disposes of a collection of 5 neural networks $\mathcal{N} = \{N^{(1)}, \ldots, N^{(5)}\}$, available at [18]. These networks all have 6 hidden layers of 50 nodes each. They yield a 5 dimensional output, on which the funtion $Post_{\text{own}} : \mathbf{y} \mapsto \pi(\text{argmin}_i\ \mathbf{y}_i)$ is applied.

Controller of the intruder: in the case (o+i), the controller of the intruder is the same as the one of the ownship, except that the Pre_{int} function returns the cylindrical coordinates of the ownship relative to intruder.

In the case (o+i), the product of the two CPSs has $5^2 = 25$ possible commands.

It is important noting that VENMAS could not be evaluated on this use case as the necessary assumptions for its use are not satisfied: nor the function f neither the function *Pre* has the correct form. We also did not evaluate SAMBA (zono), given the better performance of SAMBA (polytope) on the VCAS use case. In the following, SAMBA (polytope) is referred to as SAMBA.

Verification problems: the unsafe states are the cases where the two aircraft collide *i.e.,* the cases where the distance between the two aircraft is less than 500.0ft. The problems of type A consist of an uncertainty of ±10ft on $x_{\text{int},0}$ and $y_{\text{int},0}$, the problems of type B consist of an uncertainty of $\pm 0.1^\circ$ on $\psi_{\text{int},0}$, and the problems of type C consist of an uncertainty of ±10ft on $x_{\text{int},0}$ and $y_{\text{int},0}$, $\pm 0.1^\circ$ on $\psi_{\text{int},0}$ and $\psi_{\text{own},0}$, and ±1ft/s on $v_{\text{int},0}$ and $v_{\text{own},0}$. The time horizons are $t_{\text{end}} \in \{16s, 32s, 56s\}$. Finally, we created 2 non-crititical and 5 critical verification problems, following the approach in [19], yielding a total of 63 problems.

6.2 Results

Table 3 and pages14–15 show the results. Overall, for both the case (o+i) and the case (o), SAMBA (polytope) can solve 95% of the verification problems while NNV can only solve 82% of the verification problems (essentially due to timeouts). SAMBA always solve more problems than NNV, regardless of the different criteria. Both NNV and SAMBA have more difficulties to prove problems with large time horizons and uncertainties of type C. In terms of performance, SAMBA always outperforms NNV, independently of the criteria. However, the ratio between the verification time of NNV versus the verification time of SAMBA reduces with large t_{end} and problems of type C. As an example, this ratio equals 3.3 for the case (o) and $t_{\text{end}} = 16.0$ while it equals 1.30 for the same case and $t_{\text{end}} = 56.0$.

Interpretation. Compared to the VCAS, the non linearity of the ODE makes the approximation of the dynamics more expensive. This is particularly true for NNV, as explained in section 3.2. Moreover, keeping the reachability tree small is less critical, due to the reduced number of possible commands. The trade off between a costly approximation (NNV) versus a large reachability tree (SAMBA) is always in favor of SAMBA.

7 BENCHMARK 3: CARTPOLE

7.1 Description

The cartpole consists of a pole attached to a cart moving along a frictionless track. The cart is equipped with a controller that periodically provides a translation command: either move left or move right. The goal of the controller is to keep the cart-pole balanced *i.e.,* keep θ close to zero (see figure 10).

Variables. the state of the physical system is the vector $\mathbf{x}_p = (x\ \dot{x}\ \theta\ \dot{\theta})$ where x is the position of the cart along the track (in m), $\dot{x}$ is the velocity of the cart (in m/s), θ is the angle of the pole with respect to the vertical axis (in rad) and $\dot{\theta}$ is the rotational speed of the pole, in rad/s (see Fig. 10). The actuation command is the scalar x_c that is the acceleration of the cart along the track (in m/s^2).

Flow relation: the ODE $\dot{\mathbf{x}}_p = f(\mathbf{x}_p, \mathbf{x}_c)$ describing the dynamics of the physical system is given in [20].

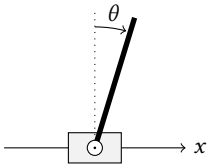


Figure 10: The Cartpole system.

Controller. the controller has a period $T = 0.02$s. It produces a command among a set of 2 possible commands. The *Pre* function is the identity function. The controller disposes of a collection of 2 neural networks, that we trained with reinforcement learning. These networks all have 2 hidden layers of 48 and 24 neurons respectively. They yield a 2 dimensional output, on which the function $Post : \mathbf{y} \mapsto \pi(\text{argmax}_i\ \mathbf{y}_i)$ is applied.

As for the ACAS, VENMAS could not be evaluated on this use case.

Verification problems: the unsafe states are the cases where the angle θ is greater than 24°. The problems of type A consist of an uncertainty of ±10cm on x, the problems of type B consist of an uncertainty of $\pm 0.1^\circ$ on θ, and the problems of type C consist of an uncertainty of ±10cm on x, ±10cm/s on $\dot{x}$, $\pm 0.1^\circ$ on θ and $\pm 0.1^\circ$/s on $\dot{\theta}$. The time horizons are $t_{\text{end}} \in \{0.2s, 0.4s, 1.0s\}$. Finally, we created 3 non-crititical and 2 critical verification problems, yielding a total of 45 problems.

7.2 Results

Table 3 and page 16 show the results. NNV can solve all the verification problems while SAMBA can only solve 49% of the problems (essentially due to false negatives). The time horizon has a significative influence on the capacity of SAMBA to solve the verification problems: SAMBA can solve all the problems where $t_{\text{end}} = 0.2$ while it can solve only 7% of the problems where $t_{\text{end}} = 1.0$, which

still represents a small time horizon. In terms of verification time, SAMBA is always faster than NNV.

Interpretation. Compared to the VCAS or the ACAS, the high complexity of the dynamics requires the solver to be able to construct a precise approximation of the reachable solutions of the ODE. SAMBA fails to build a precise enough approximation. It analyzes the system faster but its loose approximation of the dynamics leads to unknown results.

8 LESSONS LEARNED

On the applicability of formal methods. A first lesson is that formal techniques can be used for verifing the safety of real-world CPSs with controllers that are classifiers based on multiple networks: 97.3% of the totality of the 225 verification problems could be solved by at least one tool. However, this conclusion mainly applies to approximate methods. Exact methods present two issues: (1) they are not applicable to systems with non-linear dynamics and (2) they do not scale well to complex verification problems. To overcome the first issue, a possible approach could be to approximate the non-linear dynamics with a piecewise linear function and calculate the associated error, but the scalability issues would remain.

	VCAS	ACAS (o+i)	ACAS (o)	Cartpole
NNV	95.6%	98.0%	98.8%	99.9%
SAMBA (poly)	65.9%	83.6%	83.6%	97.1%

Table 4: Percentage of verification time spent on analysing the dynamics.

A second lesson regarding the applicability of formal methods is the need for an *appropriate model of the dynamics*. A same system can have its dynamics represented by different models (all correct), depending on the *coordinates* used *e.g.,* cylindrical, cartesian. Choosing the coordinates that yield the simplest model can drastically reduce the verification time, making the analysis of the dynamics easier. Indeed, as Table 4 shows, the main verification effort always lies in the analysis of the dynamics. For instance, the dynamics of the ACAS with cartesian coordinates is quite more simple than its dynamics in cylindrical coordinates. Due to the high cost of using cylindrical coordinates, no further experiments were conducted to characterize the difference of performance: an increase by a factor greater than 10 of the verification time was observed when trying to verify simple problems with such a representation.

About possible optimizations. In the case of the VCAS, a loose approximation (*e.g.,* SAMBA) yields a large reachability tree, resulting in a more expensive analysis compared to a more precise approximation (*e.g.,* NNV). To overcome this issue, two directions can be investigated: (1) improving the precision of the approximation or (2) keeping the size of the reachability tree below a certain threshold Γ. The second direction can be implemented through a heuristic that merges the closest nodes when the size of the tree exceeds the threshold Γ. We tested such a strategy on the VCAS use case, by choosing $\Gamma = 81$ (this is a heuristic choice, given that Γ needs to be greater than the number of possible commands). The results are shown in page 13. The merging strategy yields many unknown results, making the approximation too imprecise. Therefore, it appears that improving the precision would be a better direction to follow.

Choosing the best approach. Based on the experiments, we propose a heuristic for choosing the best approximate technique, depending on the CPS of interest and the nature of the verification problem:

(i) If the CPS has *many possible commands*, a precise approximation of both the neural networks (*e.g.,* star set abstraction in NNV) and the dynamics (*e.g.,* NNV) is to be preferred. Despite a costly approximation, especially for the dynamics, it would keep the reachability tree reasonably small while a loose approximation would construct a larger tree, making the overall analysis more expensive.
(ii) If the CPS has a *complex, higly non-linear dynamics*, a precise approximation of the dynamics (*e.g.,* NNV) is necessary. A loose approximation (*e.g.,* SAMBA) would not be more expensive but simply not be precise enough to achieve the proof.
(iii) In other cases, when the difficulty of the verification is more balanced between the complexity of the dynamics and the number of possible commands, a very precise approximation is not necessary. Tools such as SAMBA would allow a fast analysis while not becoming too imprecise with large time horizons.

This conclusion heavily depends on the form of the controller: without any switching mechanism between networks, building a tree would not be necessary.

About system-level verification. The last lesson is about the benefits and pitfalls of a system-level verification. Such an approach allows to circumvent the problem of the specification of neural networks. However, it appears to be quite more expensive than verifying neural networks in isolation: Table 4 shows that the time spent on analyzing the dynamics is always significantly greater than the time needed for the analysis of the neural networks.

9 CONCLUSION

This paper presented a comparative study of different formal techniques for verifying cyber-physical systems with multi-networks based classifiers as controllers. To fairly compare the different methods, we defined a unified model that expresses the verification objective as a reachability problem over a hybrid automaton. Based on three representative use cases, we could show the applicability of formal methods for this type of verification problem. We also proposed some heuristics for choosing the best method for a given problem and presented some possible optimizations.

As a future work, an interesting direction is to achieve a verification that does not assume synchronized controllers, which constitutes a common yet very restrictive hypothesis.

REFERENCES

[1] Michael E. Akintunde, Elena Botoeva, Panagiotis Kouvaros, and Alessio Lomuscio. 2020. Verifying Strategic Abilities of Neural-symbolic Multi-agent Systems. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR'20)*. 22–32.

[2] Julien Alexandre dit Sandretto and Alexandre Chapoutot. 2016. Validated Explicit and Implicit Runge-Kutta Methods. *Reliable Computing electronic edition* 22 (July 2016).

[3] Matthias Althoff. 2015. An Introduction to CORA 2015. In *ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems (EPiC Series in Computing, Vol. 34)*, Goran Frehse and Matthias Althoff (Eds.). 120–151.

[4] Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A. Henzinger, Pei-Hsin Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. 1995. The Algorithmic Analysis of Hybrid Systems. *Theoretical Computer Science B* 138 (1995), 3–34.

[5] Stanley Bak and Parasara Sridhar Duggirala. 2017. Simulation-Equivalent Reachability of Large Linear Systems with Inputs. In *Computer Aided Verification*, Rupak Majumdar and Viktor Kunčak (Eds.). Springer International Publishing, Cham, 401–420.

[6] Stanley Bak and Taylor T. Johnson. 2015. Periodically-Scheduled Controller Analysis Using Hybrid Systems Reachability and Continuization. In *2015 IEEE Real-Time Systems Symposium*. 195–205.

[7] Sergiy Bogomolov, Goran Frehse, Amit Gurung, Dongxu Li, Georg Martius, and Rajarshi Ray. 2019. Falsification of Hybrid Systems Using Symbolic Reachability and Trajectory Splicing. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC '19)*. 1–10.

[8] Mariusz Bojarski, David W. del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. 2016. End to End Learning for Self-Driving Cars. *ArXiv* abs/1604.07316 (2016).

[9] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. 2015. DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving. In *2015 IEEE International Conference on Computer Vision (ICCV)*. 2722–2730. https://doi.org/10.1109/ICCV.2015.312

[10] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. 2013. Flow*: An Analyzer for Non-linear Hybrid Systems. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 8044)*, Natasha Sharygina and Helmut Veith (Eds.). Springer, 258–263.

[11] Arthur Clavière, Eric Asselin, Christophe Garion, and Claire Pagetti. 2021. Safety Verification of Neural Network Controlled Systems. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. 47–54. https://doi.org/10.1109/DSN-W52860.2021.00019

[12] Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. 2019. Reachability Analysis for Neural Feedback Systems Using Regressive Polynomial Rule Inference. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control (HSCC 19)*. New York, NY, USA, 157–168.

[13] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. 2018. AI ^{2}: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*. 3–18.

[14] Chao Huang, Jiameng Fan, Wenchao Li, Xin Chen, and Qi Zhu. 2019. ReachNN: Reachability Analysis of Neural-Network Controlled Systems. *ACM Trans. Embed. Comput. Syst.* 18, 5s, Article 106 (Oct. 2019), 22 pages. https://doi.org/10.1145/3358228

[15] Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. 2019. Verisig: Verifying Safety Properties of Hybrid Systems with Neural Network Controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control* (Montreal, Quebec, Canada) *(HSCC 19)*. Association for Computing Machinery, New York, NY, USA, 169–178. https://doi.org/10.1145/3302504.3311806

[16] Kyle D. Julian and Mykel J. Kochenderfer. 2019. Guaranteeing Safety for Neural Network-Based Aircraft Collision Avoidance Systems. In *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*.

[17] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. 2019. Deep Neural Network Compression for Aircraft Collision Avoidance Systems. *Journal of Guidance, Control, and Dynamics* 42, 3 (2019), 598–608. https://doi.org/10.2514/1.G003724

[18] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. 2017. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In *Computer Aided Verification*, Rupak Majumdar and Viktor Kunčak (Eds.). Springer International Publishing, Cham, 97–117.

[19] Diego Manzanas Lopez, Taylor Johnson, Hoang-Dung Tran, Stanley Bak, Xin Chen, and Kerianne L. Hobbs. [n.d.]. Verification of Neural Network Compression of ACAS Xu Lookup Tables with Star Set Reachability. In *AIAA Scitech 2021 Forum*. https://doi.org/10.2514/6.2021-0995 arXiv:https://arc.aiaa.org/doi/pdf/10.2514/6.2021-0995

[20] Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Souradeep Dutta, Taylor J. Carpenter, Radoslav Ivanov, and Taylor T. Johnson. 2019. ARCH-COMP19 Category Report: Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants. In *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systemsi, part of CPS-IoT Week 2019, Montreal, QC, Canada, April 15, 2019*. 103–119.

[21] Diego Manzanas. 2020. ACAS Xu. https://github.com/mldiego/AcasXu.

[22] James D. Meiss. 2007. *Differential Dynamical Systems*. Society of Industrial and Applied Mathematics (SIAM).

[23] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos A. Theodorou, and Byron Boots. 2018. Agile Autonomous Driving using End-to-End Deep Imitation Learning. In *Robotics: Science and Systems*.

[24] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. 2019. An Abstract Domain for Certifying Neural Networks. *Proc. ACM Program. Lang.* 3, POPL (2019).

[25] Stanford Intelligent Systems Laboratory (SISL). 2020. VCAS. https://github.com/sisl/VerticalCAS.

[26] Hoang-Dung Tran, Xiaodong Yang, Diego Manzanas Lopez, Patrick Musau, Luan Viet Nguyen, Weiming Xiang, Stanley Bak, and Taylor T. Johnson. 2020. NNV: The Neural Network Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems. In *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part I*. 3–17.

[27] Hoang-Dung Tran, Diago Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. 2019. Star-Based Reachability Analysis of Deep Neural Networks. In *Formal Methods – The Next 30 Years*, Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira (Eds.). Springer International Publishing, Cham, 670–686.

[28] Hoang-Dung Tran, Weiming Xiang, and Taylor T. Johnson. 2020. Verification Approaches for Learning-Enabled Autonomous Cyber-Physical Systems. *IEEE Design Test* (2020), 1–1. https://doi.org/10.1109/MDAT.2020.3015712

[29] R.R. Zakrzewski. 2001. Fuel mass estimation in aircraft tanks using neural nets. In *Proceedings of the 40th IEEE Conference on Decision and Control (Cat. No.01CH37228)*, Vol. 4. 3728–3733 vol.4. https://doi.org/10.1109/CDC.2001.980443

A BENCHMARK RESULTS

Table 5: The table gives the verification results ("safe", "unsafe", "unknown" or "timeout") and the verification times (in s) for the VCAS benchmark. Legend: a red cell hilights an unknown verification result or a timeout while a green cell indicates the best verification time.

		VCAS benchmark							
		VENMAS		NNV		SAMBA (polytope)		SAMBA (zonotope)	
Problem		result	time	result	time	result	time	result	time
$t_{\text{end}} = 5.0$	$\mathcal{NC}_{1,A}$	safe	0.82	safe	8.32	safe	0.46	safe	0.77
	$\mathcal{NC}_{2,A}$	safe	157.78	safe	3.98	safe	0.49	safe	2.47
	$\mathcal{NC}_{1,B}$	safe	0.89	safe	4.24	safe	0.51	safe	0.84
	$\mathcal{NC}_{2,B}$	safe	0.31	safe	4.17	safe	0.49	safe	1.06
	$\mathcal{NC}_{1,C}$	safe	186.74	safe	4.26	safe	1.95	safe	4.99
	$\mathcal{NC}_{2,C}$	timeout	1h	safe	4.43	safe	12.31	timeout	1h
	$C_{1,A}$	safe	0.17	safe	4.26	safe	0.68	safe	0.99
	$C_{2,A}$	safe	0.13	safe	3.56	safe	0.86	safe	1.11
	$C_{3,A}$	safe	0.14	safe	3.68	safe	0.78	safe	0.85
	$C_{4,A}$	safe	0.13	safe	3.90	safe	0.84	safe	1.20
	$C_{1,B}$	safe	0.06	safe	3.88	safe	0.81	safe	0.80
	$C_{2,B}$	safe	0.19	safe	3.81	safe	0.86	safe	1.24
	$C_{3,B}$	safe	0.75	safe	3.67	safe	0.77	safe	0.96
	$C_{4,B}$	safe	0.18	safe	3.65	safe	1.55	safe	5.64
	$C_{1,C}$	safe	150.90	safe	4.03	safe	14.36	safe	22.01
	$C_{2,C}$	safe	288.76	safe	4.15	safe	14.41	safe	125.58
	$C_{3,C}$	safe	213.44	safe	4.39	safe	13.52	safe	21.33
	$C_{4,C}$	safe	345.62	safe	4.23	safe	13.87	safe	125.73
$t_{\text{end}} = 8.0$	$\mathcal{NC}_{1,A}$	safe	0.64	safe	5.54	safe	0.73	safe	0.82
	$\mathcal{NC}_{2,A}$	safe	0.03	safe	5.96	safe	0.75	safe	0.95
	$\mathcal{NC}_{1,B}$	timeout	1h	safe	5.94	safe	1.95	safe	3.22
	$\mathcal{NC}_{2,B}$	timeout	1h	safe	5.78	safe	0.87	safe	0.99
	$\mathcal{NC}_{1,C}$	timeout	1h	safe	6.11	safe	19.04	safe	40.76
	$\mathcal{NC}_{2,C}$	timeout	1h	safe	6.40	safe	18.19	timeout	1h
	$C_{1,A}$	timeout	1h	safe	5.46	safe	1.96	safe	3.34
	$C_{2,A}$	safe	982.62	safe	5.22	safe	1.25	safe	1.43
	$C_{3,A}$	timeout	1h	safe	5.95	safe	2.15	safe	3.54
	$C_{4,A}$	safe	570.95	safe	5.42	safe	1.15	safe	2.00
	$C_{1,B}$	safe	21.96	safe	5.65	safe	2.29	safe	3.29
	$C_{2,B}$	safe	4.13	safe	5.67	safe	1.94	safe	3.91
	$C_{3,B}$	safe	2460.15	safe	6.09	safe	2.21	safe	43.40
	$C_{4,B}$	safe	2.86	safe	5.39	safe	3.87	safe	6.32
	$C_{1,C}$	timeout	1h	safe	6.10	safe	37.57	timeout	1h
	$C_{2,C}$	timeout	1h	safe	5.79	safe	36.05	safe	82.09
	$C_{3,C}$	timeout	1h	safe	5.78	safe	37.38	timeout	1h
	$C_{4,C}$	timeout	1h	safe	5.80	safe	34.57	safe	85.11
$t_{\text{end}} = 25.0$	$\mathcal{NC}_{1,A}$	safe	0.04	safe	15.69	safe	2.43	safe	3.84
	$\mathcal{NC}_{2,A}$	timeout	1h	safe	15.54	safe	4.89	safe	6.73
	$\mathcal{NC}_{1,B}$	timeout	1h	safe	16.13	safe	2.58	safe	3.62
	$\mathcal{NC}_{2,B}$	timeout	1h	safe	17.00	safe	11.20	safe	16.12
	$\mathcal{NC}_{1,C}$	timeout	1h	safe	17.65	safe	11.08	safe	22.04
	$\mathcal{NC}_{2,C}$	timeout	1h	safe	18.35	safe	35.47	safe	46.64
	$C_{1,A}$	timeout	1h	safe	15.34	safe	2.50	safe	3.70
	$C_{2,A}$	timeout	1h	safe	15.34	safe	2.27	safe	3.73
	$C_{3,A}$	timeout	1h	safe	15.53	safe	2.58	safe	3.55
	$C_{4,A}$	timeout	1h	safe	15.49	safe	2.37	safe	3.73
	$C_{1,B}$	timeout	1h	safe	17.86	safe	15.74	safe	25.63
	$C_{2,B}$	timeout	1h	safe	16.81	safe	10.65	safe	16.24
	$C_{3,B}$	timeout	1h	safe	17.21	safe	10.83	safe	29.78
	$C_{4,B}$	timeout	1h	safe	16.86	safe	13.28	safe	12.54
	$C_{1,C}$	timeout	1h	safe	17.76	safe	656.60	timeout	1h
	$C_{2,C}$	timeout	1h	safe	18.16	safe	1529.93	timeout	1h
	$C_{3,C}$	timeout	1h	safe	18.46	safe	548.99	timeout	1h
	$C_{4,C}$	timeout	1h	safe	17.95	safe	1522.13	timeout	1h

the experiments were run on a CentOS 7 machine with 2 Intel® Xeon® processors E5-2670 v3 @ 2.30GHz of 12 cores (24 threads) each and 64 GB RAM, using the monolithic encoding for the VENMAS tool, which was shown to offer the best performance [1].

Table 6: The table gives the verification results ("safe", "unsafe", "unknown" or "timeout") and the verification times (in s) for the VCAS benchmark. Legend: a red cell hilights an unknown verification result or a timeout while a green cell indicates the best verification time.

		VCAS benchmark			
		SAMBA (polytope)		SAMBA (polytope & merge)	
Problem		result	time	result	time
$t_{\text{end}} = 5.0$	$\mathcal{NC}_{1,A}$	safe	0.46	safe	0.54
	$\mathcal{NC}_{2,A}$	safe	0.49	safe	0.63
	$\mathcal{NC}_{1,B}$	safe	0.51	safe	0.59
	$\mathcal{NC}_{2,B}$	safe	0.49	safe	0.60
	$\mathcal{NC}_{1,C}$	safe	1.95	safe	1.97
	$\mathcal{NC}_{2,C}$	safe	12.31	safe	13.47
	$\mathcal{C}_{1,A}$	safe	0.68	safe	0.79
	$\mathcal{C}_{2,A}$	safe	0.86	safe	1.05
	$\mathcal{C}_{3,A}$	safe	0.78	safe	0.83
	$\mathcal{C}_{4,A}$	safe	0.84	safe	1.02
	$\mathcal{C}_{1,B}$	safe	0.81	safe	0.76
	$\mathcal{C}_{2,B}$	safe	0.86	safe	0.95
	$\mathcal{C}_{3,B}$	safe	0.77	safe	0.75
	$\mathcal{C}_{4,B}$	safe	1.55	safe	1.62
	$\mathcal{C}_{1,C}$	safe	14.36	safe	8.87
	$\mathcal{C}_{2,C}$	safe	14.41	safe	9.08
	$\mathcal{C}_{3,C}$	safe	13.52	safe	8.95
	$\mathcal{C}_{4,C}$	safe	13.87	safe	9.23
$t_{\text{end}} = 8.0$	$\mathcal{NC}_{1,A}$	safe	0.73	safe	0.88
	$\mathcal{NC}_{2,A}$	safe	0.75	safe	0.91
	$\mathcal{NC}_{1,B}$	safe	1.95	safe	2.26
	$\mathcal{NC}_{2,B}$	safe	0.87	safe	0.84
	$\mathcal{NC}_{1,C}$	safe	19.04	safe	20.45
	$\mathcal{NC}_{2,C}$	safe	18.19	safe	20.11
	$\mathcal{C}_{1,A}$	safe	1.96	safe	2.19
	$\mathcal{C}_{2,A}$	safe	1.25	safe	1.19
	$\mathcal{C}_{3,A}$	safe	2.15	safe	2.43
	$\mathcal{C}_{4,A}$	safe	1.15	safe	1.36
	$\mathcal{C}_{1,B}$	safe	2.29	safe	2.34
	$\mathcal{C}_{2,B}$	safe	1.94	safe	2.23
	$\mathcal{C}_{3,B}$	safe	2.21	safe	2.17
	$\mathcal{C}_{4,B}$	safe	3.87	safe	4.20
	$\mathcal{C}_{1,C}$	safe	37.57	safe	34.98
	$\mathcal{C}_{2,C}$	safe	36.05	safe	36.49
	$\mathcal{C}_{3,C}$	safe	37.38	safe	33.56
	$\mathcal{C}_{4,C}$	safe	34.57	safe	34.57
$t_{\text{end}} = 25.0$	$\mathcal{NC}_{1,A}$	safe	2.43	safe	2.38
	$\mathcal{NC}_{2,A}$	safe	4.89	safe	4.63
	$\mathcal{NC}_{1,B}$	safe	2.58	safe	2.42
	$\mathcal{NC}_{2,B}$	safe	11.20	safe	11.28
	$\mathcal{NC}_{1,C}$	safe	11.08	safe	11.75
	$\mathcal{NC}_{2,C}$	safe	35.47	safe	40.23
	$\mathcal{C}_{1,A}$	safe	2.50	safe	2.61
	$\mathcal{C}_{2,A}$	safe	2.27	safe	2.65
	$\mathcal{C}_{3,A}$	safe	2.58	safe	2.65
	$\mathcal{C}_{4,A}$	safe	2.37	safe	2.17
	$\mathcal{C}_{1,B}$	safe	15.74	safe	17.07
	$\mathcal{C}_{2,B}$	safe	10.65	safe	11.94
	$\mathcal{C}_{3,B}$	safe	10.83	safe	10.53
	$\mathcal{C}_{4,B}$	safe	13.28	safe	13.83
	$\mathcal{C}_{1,C}$	safe	656.60	unknown	2474.68
	$\mathcal{C}_{2,C}$	safe	1529.93	timeout	1h
	$\mathcal{C}_{3,C}$	safe	548.99	unknown	2429.05
	$\mathcal{C}_{4,C}$	safe	1522.13	timeout	1h

the experiments were run on a CentOS 7 machine with 2 Intel® Xeon® processors E5-2670 v3 @ 2.30GHz of 12 cores (24 threads) each and 64 GB RAM, using the monolithic encoding for the samba tool, which was shown to offer the best performance [1].

Table 7: The table gives the verification results ("safe", "unsafe", "unknown" or "timeout") and the verification times (in s) for the ACAS (o+i) benchmark. Legend: a red cell hilights an unknown verification result or a timeout while a green cell indicates the best verification time.

		ACAS (o+i) benchmark			
		NNV		SAMBA (polytopes)	
Problem		result	time	result	time
$t_{\text{end}} = 16.0$	$\mathcal{NC}_{1,A}$	safe	17.35	safe	3.54
	$\mathcal{NC}_{2,A}$	safe	21.40	safe	4.75
	$\mathcal{NC}_{1,B}$	safe	12.64	safe	4.43
	$\mathcal{NC}_{2,B}$	safe	21.82	safe	4.90
	$\mathcal{NC}_{1,C}$	safe	17.75	safe	4.68
	$\mathcal{NC}_{2,C}$	safe	22.75	safe	14.13
	$C_{1,A}$	safe	16.57	safe	6.16
	$C_{2,A}$	safe	13.54	safe	5.26
	$C_{3,A}$	safe	28.42	safe	7.40
	$C_{4,A}$	safe	14.01	safe	6.64
	$C_{5,A}$	safe	23.82	safe	8.75
	$C_{1,B}$	safe	17.71	safe	6.25
	$C_{2,B}$	safe	14.50	safe	5.38
	$C_{3,B}$	safe	33.51	safe	7.27
	$C_{4,B}$	safe	14.76	safe	6.13
	$C_{5,B}$	timeout	1h	safe	41.70
	$C_{1,C}$	timeout	1h	safe	105.48
	$C_{2,C}$	timeout	1h	unknown	225.40
	$C_{3,C}$	timeout	1h	safe	121.27
	$C_{4,C}$	timeout	1h	safe	128.24
	$C_{5,C}$	timeout	1h	safe	84.52
$t_{\text{end}} = 32.0$	$\mathcal{NC}_{1,A}$	timeout	1h	safe	8.00
	$\mathcal{NC}_{2,A}$	timeout	1h	safe	9.01
	$\mathcal{NC}_{1,B}$	timeout	1h	safe	8.26
	$\mathcal{NC}_{2,B}$	timeout	1h	safe	9.31
	$\mathcal{NC}_{1,C}$	timeout	1h	safe	9.09
	$\mathcal{NC}_{2,C}$	timeout	1h	safe	34.30
	$C_{1,A}$	safe	27.51	safe	9.01
	$C_{2,A}$	safe	27.16	safe	9.36
	$C_{3,A}$	safe	28.17	safe	8.77
	$C_{4,A}$	safe	30.30	safe	8.93
	$C_{5,A}$	safe	52.32	safe	8.00
	$C_{1,B}$	safe	30.79	safe	15.21
	$C_{2,B}$	safe	28.92	safe	9.36
	$C_{3,B}$	safe	28.45	safe	9.09
	$C_{4,B}$	safe	31.23	safe	8.77
	$C_{5,B}$	timeout	1h	safe	1384.06
	$C_{1,C}$	timeout	1h	safe	45.66
	$C_{2,C}$	timeout	1h	safe	209.39
	$C_{3,C}$	timeout	1h	safe	16.12
	$C_{4,C}$	timeout	1h	safe	32.41
	$C_{5,C}$	timeout	1h	timeout	1h
$t_{\text{end}} = 56.0$	$\mathcal{NC}_{1,A}$	safe	53.54	safe	12.97
	$\mathcal{NC}_{2,A}$	safe	79.38	safe	13.95
	$\mathcal{NC}_{1,B}$	safe	51.63	safe	13.42
	$\mathcal{NC}_{2,B}$	safe	77.27	safe	14.30
	$\mathcal{NC}_{1,C}$	safe	68.92	safe	14.92
	$\mathcal{NC}_{2,C}$	safe	92.56	safe	57.84
	$C_{1,A}$	safe	55.91	safe	26.50
	$C_{2,A}$	safe	663.93	safe	23.66
	$C_{3,A}$	safe	45.24	safe	15.12
	$C_{4,A}$	safe	45.35	safe	15.01
	$C_{5,A}$	safe	85.97	safe	13.37
	$C_{1,B}$	safe	51.87	safe	48.31
	$C_{2,B}$	safe	197.20	safe	13.61
	$C_{3,B}$	safe	51.92	safe	13.98
	$C_{4,B}$	safe	50.87	safe	26.77
	$C_{5,B}$	timeout	1h	timeout	1h
	$C_{1,C}$	timeout	1h	safe	62.37
	$C_{2,C}$	timeout	1h	timeout	1h
	$C_{3,C}$	safe	124.72	safe	34.23
	$C_{4,C}$	safe	132.57	safe	93.29
	$C_{5,C}$	timeout	1h	timeout	1h

the experiments were run on a CentOS 7 machine with 2 Intel® Xeon® processors E5-2670 v3 @ 2.30GHz of 12 cores (24 threads) each and 64 GB RAM, using the monolithic encoding for the VENMAS tool, which was shown to offer the best performance [1].

Table 8: The table gives the verification results ("safe", "unsafe", "unknown" or "timeout") and the verification times (in s) for the ACAS (o) benchmark. Legend: a red cell hilights an unknown verification result or a timeout while a green cell indicates the best verification time.

		ACAS (o) benchmark			
		NNV		SAMBA (polytopes)	
Problem		result	time	result	time
$t_{\text{end}} = 16.0$	$\mathcal{NC}_{1,A}$	safe	13.23	safe	2.79
	$\mathcal{NC}_{2,A}$	safe	21.84	safe	5.43
	$\mathcal{NC}_{1,B}$	safe	9.93	safe	2.59
	$\mathcal{NC}_{2,B}$	safe	10.82	safe	2.85
	$\mathcal{NC}_{1,C}$	safe	12.27	safe	3.27
	$\mathcal{NC}_{2,C}$	safe	39.94	safe	8.08
	$C_{1,A}$	safe	10.53	safe	3.69
	$C_{2,A}$	safe	10.57	safe	3.52
	$C_{3,A}$	safe	10.31	safe	3.86
	$C_{4,A}$	safe	9.79	safe	3.19
	$C_{5,A}$	safe	10.16	safe	3.68
	$C_{1,B}$	safe	10.95	safe	3.83
	$C_{2,B}$	safe	10.75	safe	3.57
	$C_{3,B}$	safe	11.41	safe	3.47
	$C_{4,B}$	safe	11.23	safe	3.69
	$C_{5,B}$	safe	29.88	safe	6.95
	$C_{1,C}$	safe	12.57	safe	3.89
	$C_{2,C}$	safe	12.46	safe	4.16
	$C_{3,C}$	safe	12.63	safe	4.49
	$C_{4,C}$	safe	12.38	safe	4.29
	$C_{5,C}$	safe	33.87	safe	13.30
$t_{\text{end}} = 32.0$	$\mathcal{NC}_{1,A}$	safe	17.91	safe	5.75
	$\mathcal{NC}_{2,A}$	safe	42.04	safe	11.85
	$\mathcal{NC}_{1,B}$	safe	18.95	safe	5.45
	$\mathcal{NC}_{2,B}$	safe	21.33	safe	6.63
	$\mathcal{NC}_{1,C}$	safe	24.76	safe	7.54
	$\mathcal{NC}_{2,C}$	safe	75.09	safe	27.67
	$C_{1,A}$	safe	21.78	safe	7.86
	$C_{2,A}$	safe	38.82	safe	14.30
	$C_{3,A}$	safe	22.85	safe	7.89
	$C_{4,A}$	safe	22.27	safe	7.66
	$C_{5,A}$	safe	21.74	safe	7.44
	$C_{1,B}$	safe	64.37	safe	7.99
	$C_{2,B}$	safe	22.39	safe	14.73
	$C_{3,B}$	safe	22.91	safe	8.25
	$C_{4,B}$	safe	32.30	safe	10.99
	$C_{5,B}$	safe	63.60	safe	22.93
	$C_{1,C}$	safe	72.52	safe	12.36
	$C_{2,C}$	safe	115.59	safe	52.78
	$C_{3,C}$	safe	44.47	safe	13.90
	$C_{4,C}$	safe	243.16	safe	47.55
	$C_{5,C}$	safe	221.48	safe	55.87
$t_{\text{end}} = 56.0$	$\mathcal{NC}_{1,A}$	safe	33.76	safe	11.32
	$\mathcal{NC}_{2,A}$	safe	77.16	safe	23.39
	$\mathcal{NC}_{1,B}$	safe	34.06	safe	11.52
	$\mathcal{NC}_{2,B}$	safe	39.92	safe	12.93
	$\mathcal{NC}_{1,C}$	safe	43.22	safe	13.29
	$\mathcal{NC}_{2,C}$	safe	134.47	safe	51.45
	$C_{1,A}$	safe	37.01	safe	12.12
	$C_{2,A}$	safe	37.27	safe	32.61
	$C_{3,A}$	safe	37.22	safe	13.31
	$C_{4,A}$	safe	37.31	safe	12.83
	$C_{5,A}$	safe	171.33	safe	97.79
	$C_{1,B}$	safe	38.73	safe	13.85
	$C_{2,B}$	safe	38.18	safe	33.93
	$C_{3,B}$	safe	38.55	safe	13.31
	$C_{4,B}$	safe	39.10	safe	12.94
	$C_{5,B}$	safe	444.66	safe	112.33
	$C_{1,C}$	safe	43.29	safe	14.61
	$C_{2,C}$	safe	468.94	safe	149.72
	$C_{3,C}$	safe	82.53	safe	28.20
	$C_{4,C}$	safe	106.61	safe	34.58
	$C_{5,C}$	timeout	1h	timeout	1h

the experiments were run on a CentOS 7 machine with 2 Intel® Xeon® processors E5-2670 v3 @ 2.30GHz of 12 cores (24 threads) each and 64 GB RAM, using the monolithic encoding for the VENMAS tool, which was shown to offer the best performance [1].

Table 9: The table gives the verification results ("safe", "unsafe", "unknown" or "timeout") and the verification times (in s) for the Cartpole benchmark. Legend: a red cell hilights an unknown verification result or a timeout while a green cell indicates the best verification time.

		Cartpole benchmark			
		NNV		SAMBA (polytopes)	
Problem		result	time	result	time
$t_{\text{end}} = 0.2$	$\mathcal{NC}_{1,A}$	safe	25.42	safe	2.17
	$\mathcal{NC}_{2,A}$	safe	22.11	safe	2.10
	$\mathcal{NC}_{3,A}$	safe	21.17	safe	1.88
	$\mathcal{NC}_{1,B}$	safe	18.43	safe	1.77
	$\mathcal{NC}_{2,B}$	safe	21.91	safe	1.67
	$\mathcal{NC}_{3,B}$	safe	19.72	safe	1.75
	$\mathcal{NC}_{1,C}$	safe	19.32	safe	1.84
	$\mathcal{NC}_{2,C}$	safe	18.67	safe	5.19
	$\mathcal{NC}_{3,C}$	safe	19.33	safe	1.77
	$C_{1,A}$	safe	18.19	safe	1.71
	$C_{2,A}$	safe	18.25	safe	1.67
	$C_{1,B}$	safe	18.55	safe	1.74
	$C_{2,B}$	safe	18.17	safe	1.63
	$C_{1,C}$	safe	18.72	safe	1.70
	$C_{2,C}$	safe	18.82	safe	1.90
$t_{\text{end}} = 0.5$	$\mathcal{NC}_{1,A}$	safe	39.95	unknown	2.66
	$\mathcal{NC}_{2,A}$	safe	38.75	safe	9.70
	$\mathcal{NC}_{3,A}$	safe	39.14	unknown	2.07
	$\mathcal{NC}_{1,B}$	safe	38.39	unknown	2.07
	$\mathcal{NC}_{2,B}$	safe	37.45	safe	3.76
	$\mathcal{NC}_{3,B}$	safe	38.51	unknown	2.31
	$\mathcal{NC}_{1,C}$	safe	37.48	unknown	2.16
	$\mathcal{NC}_{2,C}$	safe	35.77	safe	19.50
	$\mathcal{NC}_{3,C}$	safe	35.82	unknown	2.15
	$C_{1,A}$	safe	35.70	unknown	3.14
	$C_{2,A}$	safe	36.62	safe	3.95
	$C_{1,B}$	safe	36.87	unknown	2.69
	$C_{2,B}$	safe	36.77	safe	3.48
	$C_{1,C}$	safe	36.79	unknown	2.66
	$C_{2,C}$	safe	36.75	safe	6.50
$t_{\text{end}} = 1.0$	$\mathcal{NC}_{1,A}$	safe	94.09	unknown	2.55
	$\mathcal{NC}_{2,A}$	safe	93.53	unknown	21.98
	$\mathcal{NC}_{3,A}$	safe	94.28	unknown	2.21
	$\mathcal{NC}_{1,B}$	safe	94.78	unknown	2.32
	$\mathcal{NC}_{2,B}$	safe	91.40	safe	12.14
	$\mathcal{NC}_{3,B}$	safe	91.04	unknown	2.12
	$\mathcal{NC}_{1,C}$	safe	92.71	unknown	2.14
	$\mathcal{NC}_{2,C}$	safe	91.39	unknown	56.06
	$\mathcal{NC}_{3,C}$	safe	92.20	unknown	2.42
	$C_{1,A}$	safe	89.13	unknown	2.94
	$C_{2,A}$	safe	93.52	unknown	14.01
	$C_{1,B}$	safe	119.63	unknown	2.12
	$C_{2,B}$	safe	91.17	unknown	6.85
	$C_{1,C}$	safe	91.36	unknown	2.47
	$C_{2,C}$	safe	118.97	unknown	46.88

the experiments were run on a CentOS 7 machine with 2 Intel® Xeon® processors E5-2670 v3 @ 2.30GHz of 12 cores (24 threads) each and 64 GB RAM, using the monolithic encoding for the VENMAS tool, which was shown to offer the best performance [1].